Tampere University

Khoa Nguyen

Sequence Temporal Sub-Sampling for Automated Audio Captioning

Faculty of Information Technology and Communication Sciences (ITC)
Bachelor's thesis
May 2020

Abstract

Khoa Nguyen: Sequence Temporal Sub-Sampling for Automated Audio Captioning
Bachelor's thesis
Tampere University
Bachelor's Degree Programme in Information Technology
July 2020

Audio captioning is a novel task in machine learning which involves the generation of textual description for an audio signal. For example, a method for audio captioning must be able to generate descriptions like "two people talking about football", or "college clock striking" from the corresponding audio signals. Audio captioning is one of the tasks in the Detection and Classification of Acoustic Scenes and Events 2020 (DCASE2020). Most audio captioning methods use the encoder-decoder deep neural networks architecture as a function to map the extracted features from input audio sequence to the output captions. However, the length of an output caption is considerably less than the length of an input audio signal, for example, 10 words versus 2000 audio feature vectors. This thesis work reports an attempt to take advantage of this difference in length by employing temporal sub-sampling in the encoder-decoder neural networks. The method is evaluated using the Clotho audio captioning dataset and the DCASE2020 evaluation metrics. Experimental results show that temporal sequence sub-sampling is able to improve all considered metrics, as well as memory and time complexity while training and calculating predicted output.

Keywords: audio captioning, recurrent neural networks, RNNs, temporal sequence sub-sampling.

Contents

1 Introduction

Automated audio captioning (AAC) can be defined as an inter-modal translation task, where the input to the system is an audio signal, and the output of the system is a caption of that signal using natural language processing [1]. An example of an audio captioning system is illustrated in Figure 1.1. Compared to automatic speech recognition, which transforms speech into text, AAC converts sounds from various environments into textual descriptions. Audio captioning offers an opportunity to develop methods that can learn complex information from audio data, such as spatio-temporal relationships, foreground-background differentiation, and other higher-level and abstract knowledge. Therefore, an audio captioning system can be used in various applications, ranging from machine audio understanding to human-to-machine, or machine-to-machine interaction.

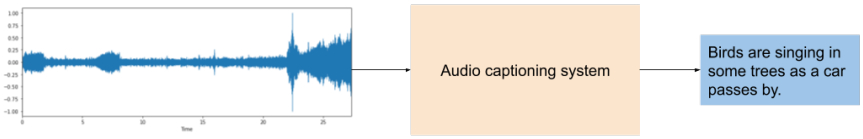

Figure 1.1 *Illustration of automated audio captioning system and process.*

Audio captioning is one of the tasks in the DCASE2020 challenge [1]. The challenge's baseline method employs an RNNs-based encoder-decoder deep neural networks (DNNs) architecture. The encoder takes a set of feature vectors, converts them into an intermediate-form representation, then the decoder uses these information to produce the final captions. Both encoder and decoder are based on recurrent neural networks with gated recurrent units. It is observed that there is a significant difference between the length of an input sequence to the neural networks and its output sequence, which indicates that one word in the output caption corresponds to multiple time steps in the input sequence. The author of this thesis hypothesizes that by reducing this length difference, the performance of a DNNs-based audio captioning method could be enhanced. The neural networks architecture in this thesis work is built upon the baseline method of the DCASE2020 audio captioning challenge as a means to test this hypothesis. More specifically, the temporal sequence sub-sampling is employed in the multi-layered encoder of the neural networks in a hierarchical way. Experiments show that temporal sequence sub-sampling could improve the metrics, while reduces the time and memory needed to train the networks as well as doing inference on unseen data. These results are published as a

[1] `http://dcase.community/challenge2020/task-automatic-audio-captioning`

workshop paper in DCASE2020 [2]. The code implementation for the method is freely available online [2].

The rest of this thesis is organized as follows: chapter 2 provides the theoretical background of all employed concepts and techniques. Chapter 3 presents the use of temporal sequence sub-sampling in the neural networks method. Chapter 4 and 5 show the procedures to evaluate the proposed method and the experimental results. Chapter 6 concludes the work and also provides some future directions for research.

[2] https://github.com/DK-Nguyen/audio-captioning-sub-sampling

2 Background

2.1 Features extraction

In general, signal processing is the term used to describe the field of science that aims to analyse time-varying physical processes. Digital signal processing describes the task of performing signal processing operations using computers or digital signal processors [3]. Digital signals are a sequence of numbers which represents discrete samples of a continuous signal in a domain, such as time or frequency. Figure 2.1 shows a continuous signal and its discrete-time representation. Audio signal processing refers to the branch of signal processing that deals particularly with audio signals [4]. The magnitude of a signal measures how far its quantity differs from zero. The power of a signal is the square of its magnitude. For the signal $x(n)$ in Figure 2.1(b), $|x(1)|$ is the magnitude at discrete-time index 1, and $|x_{pwr}(1)|^2$ is its power [3].

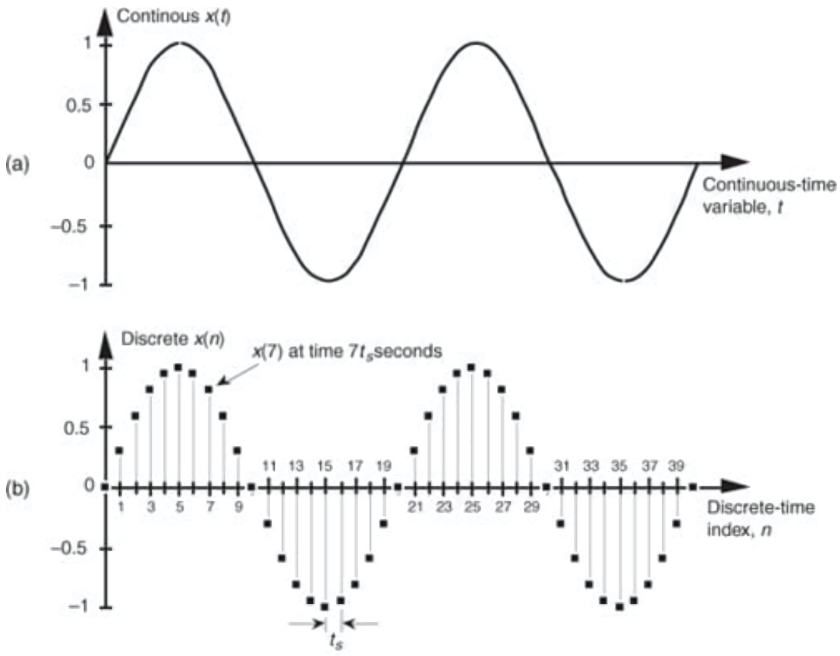

Figure 2.1 *A time-domain sine wave in* [3]*:*
(a) continuous waveform representation;
(b) discrete sample representation, we can see that the sine wave repeats every 20 samples. t_s is called sample period. If t_s is, for example, 0.05 milliseconds/sample, then the period of the sine wave is $T_{sw} = 20 \times 0.05 = 1$ millisecond*, the absolute frequency of the sine wave is $f_{sw} = 1/(1 \text{ ms}) = 1$* kHz*, and the sampling frequency (also called sample rate) is $f_s = 1/t_s = 1/(0.05 \text{ ms}) = 20000$* Hz $= 20$ kHz*. Therefore, the sine wave's absolute frequency is dependent on the sampling frequency.*

In digital signal processing, it is often necessary to represent the frequency con-

tent, also called spectral content, of discrete signals. This frequency representation is called the frequency domain. The frequency domain of a signal provides information on which part of the signal belongs to a particular frequency band over a range of different frequencies. Figure 2.2 illustrates the time- and frequency-domain representations of three signals.

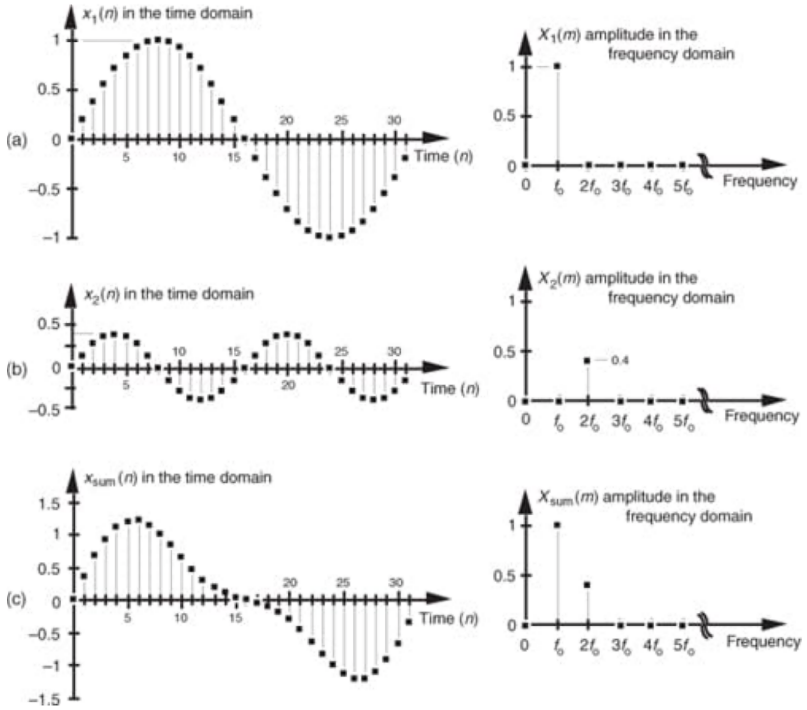

Figure 2.2 *Time- and frequency-domain representations of three related signals* [3]*:*
(a) Sine wave $x_1(n) = sin(2\pi f_0 n t_s)$ *and its spectral content* $X_1(m)$*.*
(b) On the left is the sine wave $x_2(n) = 0.4 \cdot sin(2\pi 2 f_0 n t_s)$*, whose amplitude is 0.4 of* $x_1(n)$*'s amplitude and frequency is 2 times higher than* $x_1(n)$*'s frequency. On the right is is spectral content* $X_2(m)$*.*
(c) $x_{sum}(n) = x_1(n) + x_2(n)$*. Notice from the figure of* $X_{sum}(m)$ *that* $x_{sum}(n)$ *has both the frequency* f_0 *and the frequency* $2f_0$ *with smaller amplitude.*

2.1.1 Discrete Fourier Transform

The Discrete Fourier transform (DFT) is a powerful algorithm to analyze and manipulate a discrete signal by determining its harmonic frequency content [3]. The origin of DFT is the continuous Fourier transform $X(f)$ [5], defined as

$$X(f) = \int_{-\infty}^{\infty} x(t) e^{-j2\pi ft} dt. \tag{2.1}$$

The advancements of computers and digital processors led to the development of the DFT, defined as a sequence of samples in frequency-domain $X(m)$ [3], where

$$X(m) = \sum_{n=0}^{N-1} x(n) e^{-j2\pi nm/N}. \tag{2.2}$$

The DFT equation could also be expressed in the rectangular form [3] as

$$X(m) = \sum_{n=0}^{N-1} x(n)[cos(2\pi nm/N) - jsin(2\pi nm/N)], \tag{2.3}$$

where $x(n)$ is the sequence of input samples in time domain indexed by n ($n = 0, 1, 2, 3, ..., N-1$), and $X(m)$ is the m^{th} DFT output component, m is the index of the DFT output in the frequency domain ($m = 0, 1, 2, 3, ..., N-1$).

2.1.2 Short-time Fourier Transform

In 1946, Dennis Gabor introduced the short-time Fourier transform (STFT), a technique that helps determine the phase content and the sinusoidal frequency for local sections of a signal as it changes over time [4]. Instead of producing the frequency content over the entire signal, STFT applies a windowing function only on a small section of the signal, computes a Fourier transform for the windowed signal, then shifts this process across the entire time domain representation of the signal.

Given a signal $x : \mathbb{Z} \to \mathbb{R}$, a window function $w : [0 : N-1] \to \mathbb{R}$ of length $N \in \mathbb{N}$, the hop size $H \in \mathbb{N}$ which specifies the step size of the window function, the discrete STFT $\mathcal{X}$ of x [4] is given by

$$\mathcal{X}(m, k) = \sum_{n=0}^{N-1} x(n + mH) w(n) e^{-2\pi ikn/N}, \tag{2.4}$$

where $m \in \mathbb{Z}$, and $k \in [0 : \frac{N}{2}]$. The complex value $\mathcal{X}(m, k) \in \mathbb{Z}$ indicates the k^{th} Fourier coefficient of the m^{th} time frame. The spectrogram of x is the squared magnitude of the STFT, denoted as $\mathcal{Y}(m, k)$ [4] and is calculated by

$$\mathcal{Y}(m, k) = |\mathcal{X}(m, k)|^2. \tag{2.5}$$

The spectrogram is often visualized by a 3D plot [4], where the x-axis represents the time, the y-axis represents the frequency, and the color shows the intensity of the spectrogram. More specifically, the color shows the value of the spectrogram $\mathcal{Y}(m, k)$ at the time frame indexed by m and the frequency indexed by k.

2.1.3 Mel-band energies

The human hearing perception does not follow a linear scale [6], therefore, transforming the audio signal into mel-band energies would allow the signal processing system to mimic this feature of the human auditory system [7]. The mel-band energies of an audio signal is achieved by applying the mel-scaled filter, shown in Figure 2.3, on the spectrogram of the signal. The mel scale is calculated as [8]

$$Mel(f) = 2595\ log_{10}(1 + \frac{f}{700}). \tag{2.6}$$

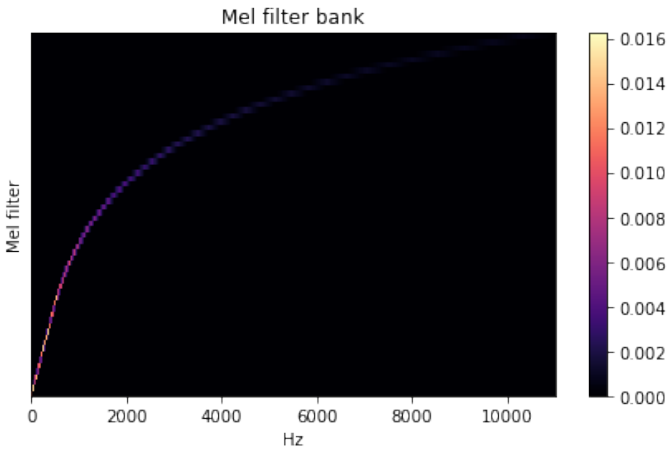

Figure 2.3 *Mel-scaled filter.*

2.2 Neural Networks

2.2.1 Feedforward Neural Networks

Feedforward neural networks (FNNs) is an algorithm with the goal to approximate some function, for example, a classifier $f(\mathbf{x})$ that tries to map a given input vector $\mathbf{x}$ to a category y. If $f(\mathbf{x})$ is parameterized by a set of parameters $\boldsymbol{\theta}$, then a FNNs can be used to define a mapping $\hat{y} = f(\mathbf{x}, \boldsymbol{\theta})$ that uses an optimizer to learn and adapt the parameters $\boldsymbol{\theta}$ for producing the best function approximation [9]. In other words, if we have a function $\mathcal{L}$ that calculates how close the predicted output $\hat{y}$ is to the ground truth value y, the optimizer of the neural networks should be able to find a set of parameters $\boldsymbol{\theta}$ that minimizes $\mathcal{L}$. $\mathcal{L}$ is often referred to as the loss function, or cost function. Selecting an optimizer and a loss function is essential in constructing a DNNs [10]. Some examples of an optimizer for FNNs is ADAM [11], MOMENTUM [12], or ADAGRAD [13]. There are many kinds of loss functions [14],

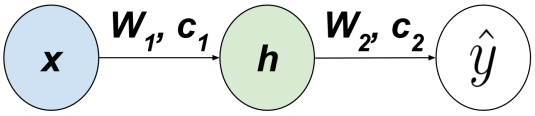

Figure 2.4 *A simple feedforward network with two layers.*

for example, the mean square error is $\mathcal{L}_{MSE} = (y - \hat{y})^2$. In Figure 2.4, we have an example of a simple FNNs, which accepts an input vector $\mathbf{x}$ and produces a predicted value $\hat{y}$. The predicted output $\hat{y}$ is not calculated directly from the input $\mathbf{x}$, but through an intermediate step called hidden unit $\mathbf{h}$, and

$$\mathbf{h} = f^{(1)}(\mathbf{x}; \mathbf{W}_1, \mathbf{c}_1). \tag{2.7}$$

Most neural networks calculate the hidden units using an affine transformation controlled by a set of learned parameters, in this case $\mathbf{W}_1$ and $\mathbf{c}_1$, followed by a nonlinear function called an activation function. Therefore, $\mathbf{h}$ is defined as

$$\mathbf{h} = f^{(1)}(\mathbf{x}; \mathbf{W}, \mathbf{c}) = g(\mathbf{W}_1\mathbf{x} + \mathbf{c}_1), \tag{2.8}$$

where g is the activation function; some popular options for g are Sigmoid, Hyperbolic Tangent (Tanh), Softmax, or Rectified Linear Unit (ReLU) [15], [16]. The approximated output $\hat{y}$ is then

$$\hat{y} = f^{(2)}(\mathbf{h}; \mathbf{W}_2, \mathbf{c}_2). \tag{2.9}$$

The complete feedforward network, therefore, is

$$\hat{y} = f(\mathbf{x}, \boldsymbol{\theta}) = f(\mathbf{x}; \mathbf{W}_1, \mathbf{c}_1, \mathbf{W}_2, \mathbf{c}_2) = f^{(2)}(f^{(1)}(\mathbf{x})), \tag{2.10}$$

where $\{\mathbf{W}_1, \mathbf{W}_2\}$ are called weights, and $\{\mathbf{c}_1, \mathbf{c}_2\}$ are called biases. We can see that the function $f(\mathbf{x}, \boldsymbol{\theta})$ is constructed by a chain of functions, such that

$$f(\mathbf{x}, \boldsymbol{\theta}) = f^{(2)}(f^{(1)}(\mathbf{x}, \boldsymbol{\theta}_1), \boldsymbol{\theta}_2). \tag{2.11}$$

This way, $f^{(1)}$ is called the first layer of the network with its parameters $\boldsymbol{\theta_1} = \{\mathbf{W}_1, \mathbf{c}_1\}$, and $f^{(2)}$ is the second layer with its parameters $\boldsymbol{\theta_2} = \{\mathbf{W}_2, \mathbf{c}_2\}$. The length of the functions chain is the depth of the neural networks. By stacking different layers on top of each other, the FNNs can learn to approximate more complex functions by composing several simpler functions together [9]. Empirical results show that greater depth produces better generalization of the neural networks on unseen data for various tasks [17]–[19].

The process of calculating a predicted output $\hat{y}$ from the input $\mathbf{x}$ is called forward propagation [9]. It is important that in the first forward propagation, the parameters of the networks are carefully initialized. This is an active research area, and many weight initialization methods [20]–[22] have been proposed. Once the value of $\hat{y}$ is calculated, the loss function $\mathcal{L}$ can be computed from $\hat{y}$ and y, the parameters $\boldsymbol{\theta}$ are updated using gradient descent and backpropagation algorithms, which will be explained in more details in section 2.2.2.

2.2.2 Backpropagation Algorithm

As $\mathcal{L}$ is computed from y and $\hat{y} = f(\mathbf{x}, \boldsymbol{\theta})$, it can be parameterized as $\mathcal{L}(\boldsymbol{\theta}, \mathbf{x}, y)$, and each parameter can be seen as a dimension in the loss function's input space. In general, because most feedforward neural networks try to approximate high dimensional nonlinear functions, the loss function becomes non-convex [9]. Therefore, to minimize the loss function, the parameters $\boldsymbol{\theta}$ are often updated through an iterative optimization algorithm called gradient descent, using the gradient of the cost function with respect the parameters, i.e. $\nabla_{\boldsymbol{\theta}}\mathcal{L} = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}}$. Backpropagation algorithm is the process of calculating the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}$ [23]. Theoretically, if there exists a non-zero gradient of a function at a point p, then the value of the function at p increases most quickly along the direction of that gradient [24]. By moving the parameters $\boldsymbol{\theta}$ to the opposite direction of the gradient $\nabla_{\boldsymbol{\theta}}\mathcal{L}$, we can minimize the loss function $\mathcal{L}(\boldsymbol{\theta}, \mathbf{x}, y)$ [25]. Figure 2.5 shows the information flow of the backpropagation process. In the figure, we denote $D_{f_i} = \frac{\partial \mathcal{L}}{\partial f_i}$, and $\nabla \boldsymbol{\theta}_i = \frac{\partial \mathcal{L}}{\partial \boldsymbol{\theta}_i}$. By chain rule of calculus [9], we can observe that

$$\nabla \boldsymbol{\theta}_i = D_{f_i} \cdot \frac{\partial f_i}{\partial \boldsymbol{\theta}_i}. \tag{2.12}$$

In the case that we do not only have one training example $\mathbf{x}$, but a dataset of many training examples $\mathbf{x}^{(i)}$, $i = \{1, 2, ..., m\}$. Then, there are three types of gradient descent which can be performed on this dataset: batch gradient descent, stochastic gradient descent, and mini-batch gradient descent [25].

- Batch gradient descent computes the gradient and update the parameters for the entire dataset in one run: $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \cdot \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta})$. Here, α is the learning rate that defines how large the parameters would be updated for each step. Furthermore, the term batch size refers to the number of training examples used in one update iteration, in this case, batch size is m.

- Stochastic gradient descent performs the parameters update for each training example $\mathbf{x}^{(i)}$ and each ground truth output y: $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \cdot \nabla_{\boldsymbol{\theta}}\mathcal{L}(\boldsymbol{\theta}, \mathbf{x}^{(i)}, y)$. For stochastic gradient descent, batch size is 1.

- Mini-batch gradient descent updates the neural networks' parameters for every mini-batch of n training examples: $\boldsymbol{\theta} = \boldsymbol{\theta} - \alpha \cdot \nabla_{\boldsymbol{\theta}} \mathcal{L}(\boldsymbol{\theta}, \mathbf{x}^{(i:i+n)}, y)$. Here, batch size is n. Mini-batch gradient descent is used most often in practice.

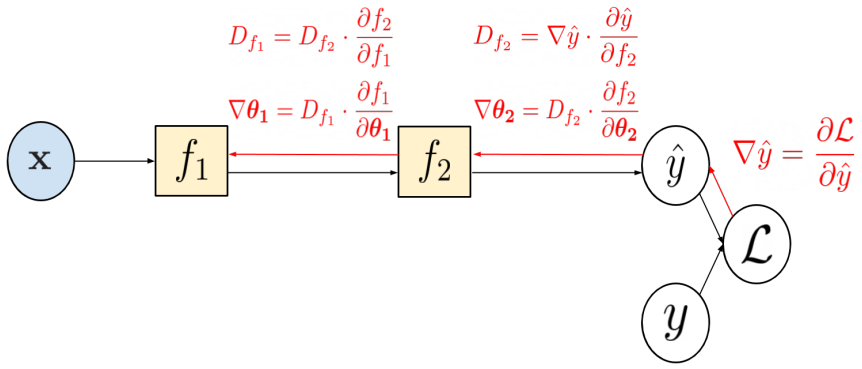

Figure 2.5 *The information flow in backpropagation (shown by red arrows) of the FNNs in Figure 2.4.*

2.3 Recurrent Neural Networks

Recurrent neural networks (RNNs) is a special type of FNNs designed to process sequential data [9]. RNNs were introduced by Rumelhart, Hinton, and Williams in 1986 [23], but they only gained popularity in recent years thanks to the advancement of powerful computing units such as the Graphics Processing Units (GPUs) that allow calculating and optimizing RNNs possible in real-time [26]. Figure 2.6 illustrates a simple RNNs.

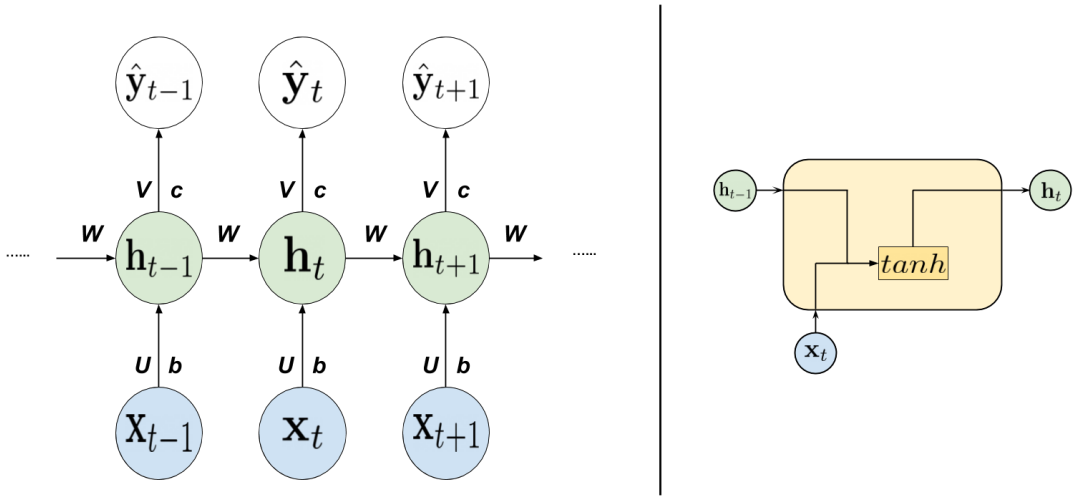

Figure 2.6 *(Left) The computational graph of a RNNs. (Right) An RNN cell, the calculation is according to Eq.* (2.13).

In Figure 2.6, the sequence $\mathbf{x}_1, ..., \mathbf{x}_{t-1}, \mathbf{x}_t, \mathbf{x}_{t+1}, ..., \mathbf{x}_T$ is the input to the RNNs. The input-to-hidden connections, such as from $\mathbf{x}_t$ to $\mathbf{h}_t$, are parameterized by a weight matrix $\mathbf{U}$ and a bias vector $\mathbf{b}$, the hidden-to-hidden connections, e.g. from $\mathbf{h}_{t-1}$ to $\mathbf{h}_t$, are parameterized by a weight matrix $\mathbf{W}$, similarly, the connections between the hidden and the output layer are parameterized by $\mathbf{V}$ and $\mathbf{c}$ [9]. By looking at the left graph in Figure 2.6, one can see that the term "recurrent" makes sense because the hidden state $\mathbf{h}_t$ refers back to the previous state $\mathbf{h}_{t-1}$. The hidden state $\mathbf{h}_t$ and the predicted output $\hat{\mathbf{y}}_t$ at time-step t are calculated as

$$\mathbf{h}_t = f(\mathbf{b} + \mathbf{U}\mathbf{x}_t + \mathbf{W}\mathbf{h}_{t-1}), \tag{2.13}$$

$$\hat{\mathbf{y}}_t = g(\mathbf{c} + \mathbf{V}\mathbf{h}_t). \tag{2.14}$$

Once the estimated sequence $\hat{\mathbf{y}}_1, ..., \hat{\mathbf{y}}_t, ..., \hat{\mathbf{y}}_T$ is computed, we can find the value of the loss function $\mathcal{L}$ by comparing it with the ground truth sequence $\mathbf{y}_1, ..., \mathbf{y}_t, ...,$ $\mathbf{y}_T$. Generally, computing the gradient of the RNNs is similar to the backpropagation algorithm described in section 2.2.2 [9].

2.3.1 Bidirectional Recurrent Neural Networks

From section 2.3, we could see that the predicted output $\hat{\mathbf{y}}_t$ is calculated based only on the information captured from the past, i.e. from $\mathbf{x}_1, ..., \mathbf{x}_{t-1}$. However, what if we want to calculate $\hat{\mathbf{y}}_t$ with the information captured from the entire input sequence? This is the reason why the Bidirectional RNNs (Bi-RNNs) were invented [27].

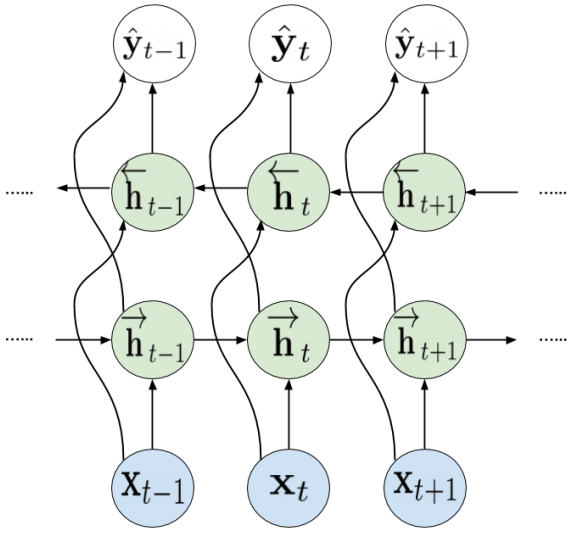

Figure 2.7 *Computational graph of a simple Bi-RNNs.*

Figure 2.7 shows a simple Bi-RNNs, a combination of an RNN that travels forward in time from the start to the end of the input sequence, and an RNN that travels backward in time from the end to the start of the input sequence. $\overrightarrow{\mathbf{h}}_t$ represents the hidden unit of the forward sub-RNN at time t, while $\overleftarrow{\mathbf{h}}_t$ is the hidden unit of the backward sub-RNN. This way, a predicted output $\hat{\mathbf{y}}_t$ captures both information from the "past" and the "future" of the input sequence [9]. Bi-RNNs have been proven to be very successful in many applications, such as speech recognition [28], [29], or handwriting recognition [30]–[32].

2.3.2 Gated Recurrent Unit

One of the appeals of RNNs is that they could incorporate previous information to solve the present task. For example, consider a language model using RNNs to predict the next word based on previous ones in a sentence [33], [34]. If the model is trying to predict the last word of "the dog is *barking*", then it is quite obvious that the word *barking* is going to be highly probable, because it stands close to the word "dog". However, what if the model tries to predict the last word of "I visited Vietnam last summer, I could talk to the people there because I knew a bit of *Vietnamese*". In this case, the model is dependent on the word "Vietnam", which lies much further back in the sentence, to predict the last word "Vietnamese". Unfortunately, when the gap between relevant information grows, RNNs become unreliable to learn them, which is referred to as long-term dependencies and was explored in detail in [35], [36]. The general idea is that the gradients tend to either vanish or explode, when they propagate over many stages [37], [38].

In order to solve this problem, the long short-term memory (LSTM) was introduced, which helps the neural networks to accumulate information over a long period, then learn to "forget" these information after they have been used [39]. The gated recurrent units (GRUs) is a modified, simpler version of LSTM, first introduced in [33]. Instead of taking only an input vector and a previous hidden state, each LSTM or GRUs cell controls its own memory and what to forget using "gates". A GRU cell consists of three different gates: the reset gate $\mathbf{r}_t$, the update gate $\mathbf{z}_t$, and the new gate $\mathbf{n}_t$ [33]. These gates are illustrated in Figure 2.8, and are calculated as

$$\mathbf{r}_t = \sigma(\mathbf{W}_{ir}\mathbf{x}_t + \mathbf{b}_{ir} + \mathbf{W}_{hr}\mathbf{h}_{t-1} + \mathbf{b}_{hr}), \tag{2.15}$$

$$\mathbf{z}_t = \sigma(\mathbf{W}_{iz}\mathbf{x}_t + \mathbf{b}_{iz} + \mathbf{W}_{hz}\mathbf{h}_{t-1} + \mathbf{b}_{hz}), \text{ and} \tag{2.16}$$

$$\mathbf{n}_t = \tanh\left(\mathbf{W}_{in}\mathbf{x}_t + \mathbf{b}_{in} + \mathbf{r}_t \odot (\mathbf{W}_{hn}\mathbf{h}_{t-1} + \mathbf{b}_{hn})\right), \tag{2.17}$$

where the reset gate has its own parameters, that are the input weights $\mathbf{W}_{ir}$, the recurrent weights $\mathbf{W}_{hr}$, and the corresponding biases $\{\mathbf{b}_{ir}, \mathbf{b}_{hr}\}$ [33]. This is similar

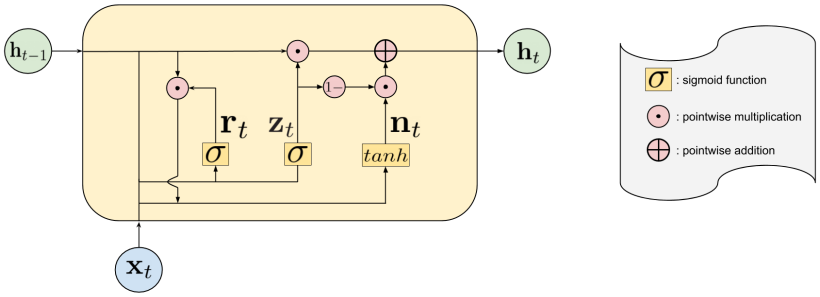

Figure 2.8 *GRU cell i at time step t.*

for the other gates. The hidden unit at time-step t is calculated based on the values of the gates by

$$\mathbf{h}_t = (1 - \mathbf{z}_t) \odot \mathbf{n}_t + \mathbf{z}_t \odot \mathbf{h}_{t-1}. \tag{2.18}$$

2.3.3 Encoder-decoder sequence-to-sequence architecture

From Figure 2.6 and Figure 2.7, we can see that the input and the output sequences of the RNNs have the same length. However, what if their lengths vary, as in the case of many applications, including audio captioning? In order to solve this problems, researchers come up with the encoder-decoder, or sequence-to-sequence, neural networks architecture [9]. The first encoder-decoder architecture, shown in

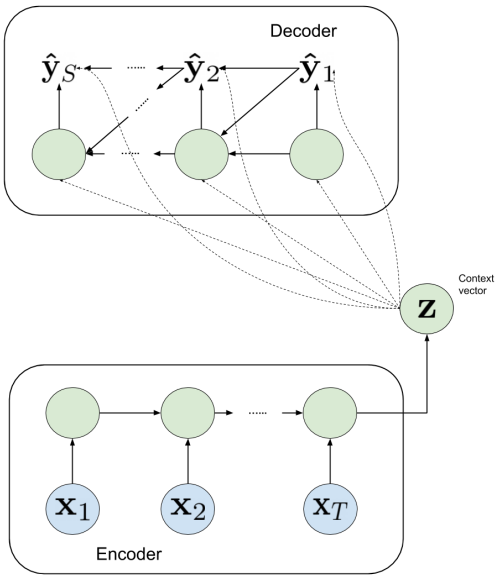

Figure 2.9 *The Encoder-Decoder Architecture used in [33].*

Figure 2.9, was introduced by Cho, Merrienboer, Gulcehre, *et al.* [33], followed

by Sutskever, Vinyals, and Le, whose method achieved state-of-the-art results for the machine translation task at that time [34]. The idea behind this architecture is that:

- First, the encoder reads and processes the input sequence $\mathbf{X} = [\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_t, ..., \mathbf{x}_T]$, then produces a vector or sequence of vectors that summarize $\mathbf{X}$. This vector is often called the context vector, denoted by $\mathbf{z}$.
- The decoder then processes the context vector to build the estimated output sequence $\hat{\mathbf{Y}} = [\hat{\mathbf{y}}_1, \hat{\mathbf{y}}_2, ..., \hat{\mathbf{y}}_s, ..., \hat{\mathbf{y}}_S]$. Note that in the context of audio captioning, the length of the input sequence (T) is often much bigger than the length of the output sequence (S).

2.3.4 Sequence Temporal Sub-sampling

In audio captioning, the input to the system is a sequence that contains thousands of audio feature vectors, and the output of the system is often a sentence of only 10 to 20 words [40]. This significant difference in lengths indicates that multiple time steps of the input feature vectors correspond to one output word in a predicted caption. Sequence temporal sub-sampling (STS) helps to take an advantage of this by removing potentially redundant or repeated input feature vectors by dropping a portion of them along the temporal axis; this property has been proven effective in other tasks, such as video classification [41].

Figure 2.10 illustrates the sub-sampling operation employed in this work, with the sub-sampling factor $M = 2$. The input matrix $\mathbf{O}$ to STS is a sequence of A vectors, each vector has B features, therefore $\mathbf{O} \in \mathbb{R}^{A \times B}$. The output matrix is $\mathbf{O}'' \in \mathbb{R}^{\lfloor A/M \rfloor \times B}$, where $\lfloor \cdot \rfloor$ denotes the floor function. The red "X"s indicates that the corresponding vectors are discarded. Here, the bigger M is, the more input vectors are discarded. In this work, STS is employed in the encoder of the sequence-to-sequence neural networks architecture. The method is explained in more detail in chapter 3.

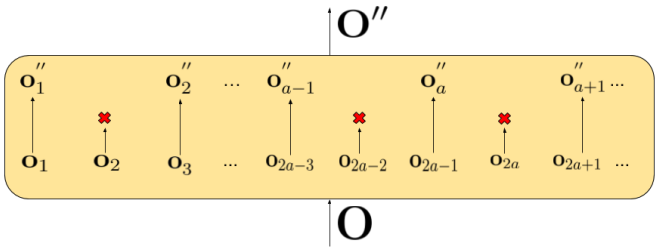

Figure 2.10 *Sequence temporal sub-sampling with factor $M = 2$.*

2.4 Related Works in Audio Captioning using Deep Neural Networks

The audio captioning task was started in 2017 by Drossos, Adavanne, and Virtanen, where they used a recurrent neural networks (RNNs) on a dataset that contains audio data and their corresponding captions extracted from the PSE library [1]. After that, they constructed Clotho - a specialized dataset for audio captioning [40], [42]. The Clotho dataset is used in the DCASE2020 audio captioning task, the challenge attracts many teams from different organizations, and all the works that achieve better metrics than the baseline method make use of deep neural networks. The final results and rankings could be found at[1]. Koizumi, Takeuchi, Ohishi, *et al.* achieve the first place of the competition, they focus on solving the two indeterminacy problems in automated audio captioning: word selection indeterminacy and sentence length indeterminacy by using data augmentation and multi-task learning [43]. The system from Wu, Chen, Wang, *et al.* ends up at the second place [44], which consists of a 10-convolutional-layer [45] encoder for audio features extraction, followed by a Transformer [46] decoder for language modeling. The convolutional encoder is pre-trained by converting the audio captioning task into a multi-label classification task, then they employed label smoothing and data augmentation during the training process of the decoder's parameters. The third place belongs to Wang, Yang, Zou1, *et al.*, where they use a convolutional neural network (CNNs) encoder and a long-short term memory (LSTM)-based decoder together with temporal attention [47]. Other works that employed the architecture with CNNs-based encoder and RNNs-based decoder, as in [48], [49], also achieve better results than the baseline method.

[1] `http://dcase.community/challenge2020/task-automatic-audio-captioning-results`

3 Method

This work employs the RNN-based encoder-decoder neural network architecture, where the input to the encoder is a sequence of T audio feature vectors with F features, $\mathbf{X} \in \mathbb{R}^{T \times F}$, and the output of the decoder is a sequence of S vectors $\hat{\mathbf{Y}} = [\hat{\mathbf{y}}_1, \ldots, \hat{\mathbf{y}}_S]$. Each vector $\hat{\mathbf{y}}_s \in [0,1]^D$ in the output contains the predicted probability for each of the D words, where D is the number of all unique words in the captions of the training dataset. The encoder is constructed in a hierarchical way with multiple RNNs-based layers, and temporal sequence sub-sampling is applied on the output of each RNN layer. The decoder consists of a RNNs layer, followed by a linear layer, according to the baseline method.

The encoder has L_{enc} bi-directional RNN layers, as illustrated in Figure 3.1. $\overrightarrow{\text{RNN}}^l_{\text{enc}}$ and $\overleftarrow{\text{RNN}}^l_{\text{enc}}$ are denoted to be the l-th layer's forward and backward RNNs. For the first layer of the encoder, the input sequence is processed as

$$\overrightarrow{\mathbf{h}}_{1_t} = \overrightarrow{\text{RNN}}^1_{\text{enc}}(\mathbf{x}_t, \overrightarrow{\mathbf{h}}_{1_{t-1}}) \text{ and} \tag{3.1}$$

$$\overleftarrow{\mathbf{h}}_{1_t} = \overleftarrow{\text{RNN}}^1_{\text{enc}}(\overleftarrow{\mathbf{x}}_t, \overleftarrow{\mathbf{h}}_{1_{t-1}}), \tag{3.2}$$

where $\mathbf{x}_t$ is the t-th feature vector in $\mathbf{X}$, $\overleftarrow{\mathbf{x}}_t$ is the time-reversed version of $\mathbf{x}_t$; and $\overrightarrow{\mathbf{h}}_{1_t}, \overleftarrow{\mathbf{h}}_{1_t} \in [-1,1]^{\Xi}$ are the outputs of the $\overrightarrow{\text{RNN}}^1_{\text{enc}}$ and $\overleftarrow{\text{RNN}}^1_{\text{enc}}$ for the time step t, respectively, $\overrightarrow{\mathbf{h}}_{1_0} = \overleftarrow{\mathbf{h}}_{1_0} = [0]^{\Xi}$. Here, Ξ is the number of output features for each of the l-th RNN layer. Then, $\overrightarrow{\mathbf{h}}_{1_t}, \overleftarrow{\mathbf{h}}_{1_t}$ are concatenated as

$$\mathbf{h}_{1_t} = [\overrightarrow{\mathbf{h}}^\top_{1_t}, \overleftarrow{\mathbf{h}}^\top_{1_t}]^\top \tag{3.3}$$

to produce the output matrix of the first layer $\mathbf{H}_1 = [\mathbf{h}_{1_1}, \ldots, \mathbf{h}_{1_T}]$.

Each layer $2 \leq l < L$ of the encoder accepts the input matrix $\mathbf{H}''_{l-1}$ as the result from applying temporal sub-sampling on the output of the previous layer

$$\mathbf{H}''_{l-1} = \{\mathbf{h}^{l-1}_{iM+1}\}^{i=\lfloor (T_l)/M \rfloor}_{i=0}, \tag{3.4}$$

where T_l is the number of time-steps of $\mathbf{H}''_{l-1}$, $M \in \mathbb{N}^\star$ is the sub-sampling factor, and $\lfloor \cdot \rfloor$ is the floor function. $\mathbf{H}'_l$ is denoted as the output from putting $\mathbf{H}''_{l-1}$ into $\overrightarrow{\text{RNN}}^l_{\text{enc}}$ and $\overleftarrow{\text{RNN}}^l_{\text{enc}}$, according to Eqs. (3.1), (3.2), and (3.3). The final output of a l-th layer with $2 \leq l < L$, $\mathbf{H}_l \in \mathbb{R}^{T_l \times \Delta}$, is attained from a residual connection between $\mathbf{H}''_{l-1}$ and $\mathbf{H}'_l$ as

$$\mathbf{H}_l = \mathbf{H}'_l + \mathbf{H}''_{l-1}. \tag{3.5}$$

By using Eq. (3.4), the encoder is forced to squeeze the information from the in-

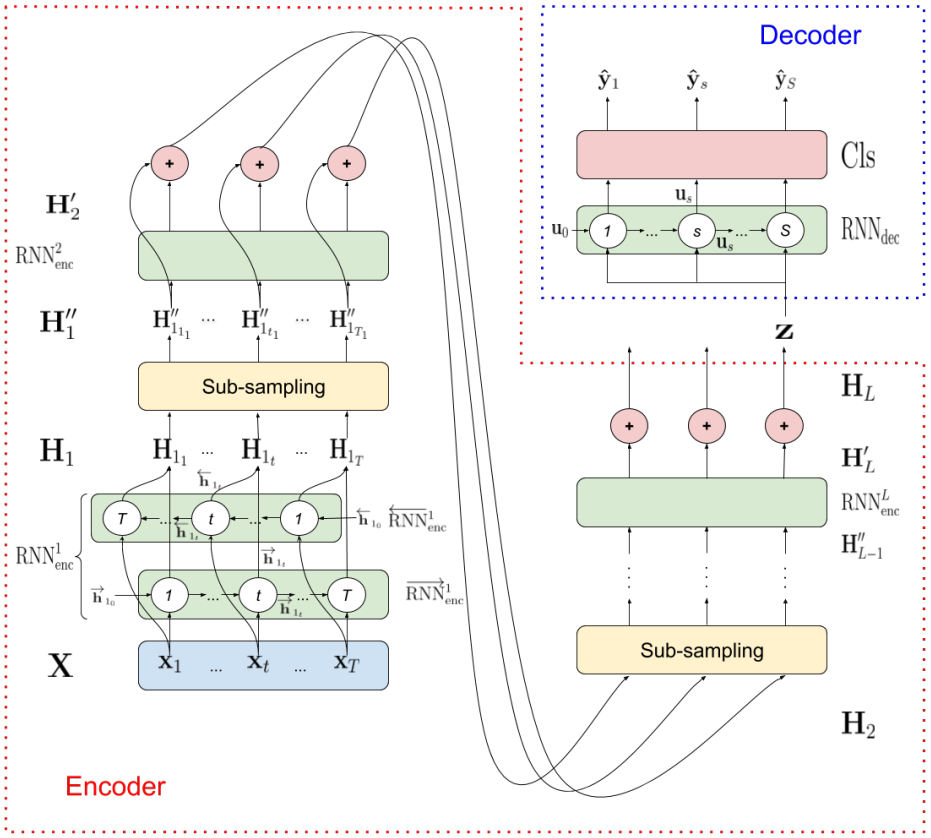

Figure 3.1 *RNNs with Temporal Sequence sub-sampling Encoder.*

put sequence to a smaller output sequence, in other words, it has to learn with lower temporal resolution. This process could make the RNNs to learn a time-filtering and time-compression of the information in the input sequence, according to [41], [50]. This has been proven to be beneficial in the case of audio captioning, where there is a significant difference between the lengths of input and output sequences [2].

The output of the encoder is $\mathbf{H}_L = [\mathbf{h}_{L_1}, \ldots, \mathbf{h}_{L_{T_L}}]$. The decoder then takes the last time-step of the L-th bi-directional RNN layer of the encoder input, so

$$\mathbf{z} = \mathbf{h}_{L_{T_L}}, \tag{3.6}$$

and $\mathbf{z} \in \mathbb{R}^{\Delta}$. As discussed in section 2.3.3, $\mathbf{z}$ is also called the fixed context vector. The decoder consists of a RNN layer, followed by a linear layer with the softmax activation function called the classifier. The output at the s-th time-step of the $\mathrm{RNN}_{\mathrm{dec}}$ is computed as

$$\mathbf{u}_s = \mathrm{RNN}_{\mathrm{dec}}(\mathbf{z}, \mathbf{u}_{s-1}), \tag{3.7}$$

where $\mathbf{u}_s \in [0,1]^{\Psi}$, $\mathbf{u}_0 = [0]^{\Psi}$. The predicted output vectors $\hat{\mathbf{y}}_s$ is then calculated by

$$\hat{\mathbf{y}}_s = \text{Cls}(\mathbf{u}_s), \tag{3.8}$$

repeatedly until $s = S$. The predicted word at a time-step s is obtained by getting the word with the highest probability in $\hat{\mathbf{y}}_s$. The encoder, the decoder, and the classifier are jointly optimized in order to minimize the binary cross-entropy loss between the predicted, $\hat{\mathbf{y}}_s$, and ground truth one-hot encoding of words, $\mathbf{y}_s = [y_{s,1}, \ldots, y_{s,D}]$.

4 Evaluation

4.1 Dataset pre-processing

In order to evaluate the proposed method, the Clotho dataset [40] is used. Clotho is built to offer diverse audio content and corresponding captions, with careful eliminations of spelling errors and named entities. The audio samples of Clotho are collected from the Freesound platform [51], together with the tags that indicate various topics of the audios. The corresponding captions of the audio samples are annotated using the Amazon Mechanical Turk (AMT) service [52] by employing a three-step based framework: audio description, description editing, description scoring [42]. The whole dataset contains 4981 audio samples of duration from 15 to 30 seconds with CD-quality of 44.1 kHz sampling rate, 16-bit sample width. Each audio sample has 5 corresponding captions with lengths ranging from 8 to 20 words, resulting in 24905 captions. The dataset is then split into three subsets, namely development, evaluation, and testing according to the 60%, 20%, 20% ratio, using multi-label stratification [53]. This process results in 2893 audio samples and 14465 captions for the development split, 1045 audio samples and 5225 captions for the evaluation split, 1043 audio samples and 5215 captions for the testing split. The development and evaluation splits are freely distributed on the Zenodo platform [1]. In this work, the development split is used to train the neural networks and fine tune its parameters, and the evaluation split is used to evaluate the metrics.

Given the Clotho dataset, the audio samples and their corresponding captions are further processed as described in the stages below in order to be used in the DNNs method:

Stage 1 - Processing of the audio samples as the input to the DNNs: from each audio sample, a power spectrogram is calculated using the short-time Fourier transform algorithm, with a Hanning windowing function of 1024-sample long ($\approx$ 23 ms) and 50% overlap. After that, a $F = 64$ log-scaled mel-band energies is extracted using a mel-scaled filter and the log operation. The log mel-band energies are used instead of mel-band because they have better properties, for example, better distribution of values. This process, as illustrated in Figure 4.1, results in a feature matrix $\mathbf{X} \in \mathbb{R}^{T \times 64}$, with $T \in [1292, 2584]$ since the audio samples are 15s and 30s long. $\mathbf{X}$ is used as the input for the neural networks described in chapter 3.

Stage 2 - As stated in [40], the ground truth captions of Clotho are processed by the following steps: punctuation removal; transforming all letters to small case; assigning a unique index for each word (tokenization); employing the start and end

[1] `https://zenodo.org/record/3490684`

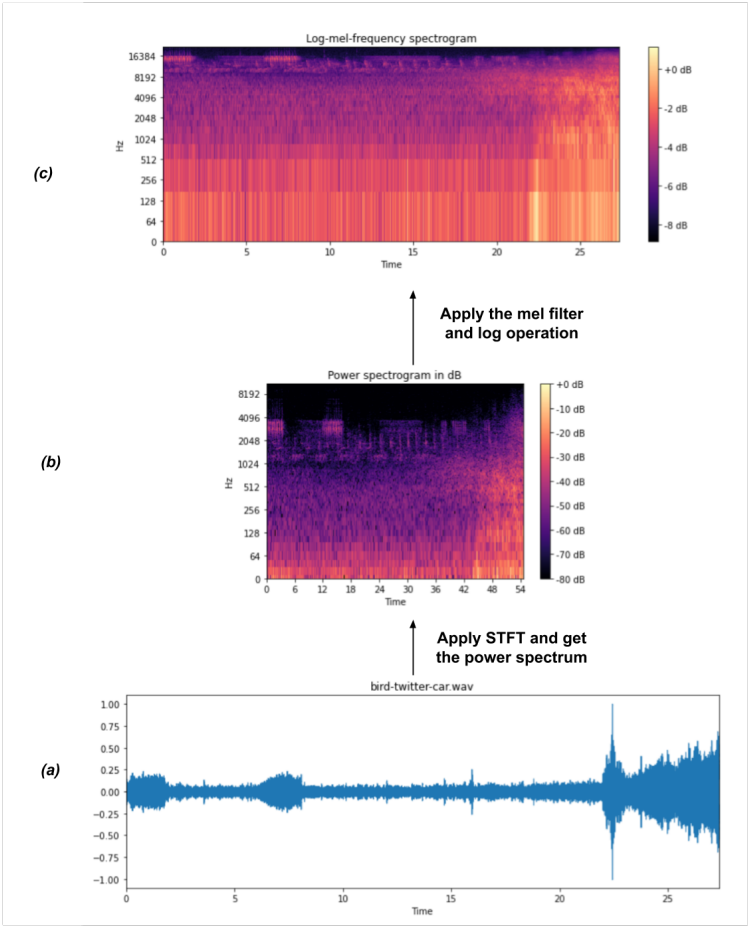

Figure 4.1 *Audio features extraction pipeline.*
(a) *Raw audio waveform.* ***(b)*** *Spectrogram.* ***(c)*** *Log mel-band energies.*

of sequence token, i.e. ⟨sos⟩ and ⟨eos⟩. This process results in a sequence of $S = 8$ to $S = 22$ one-hot encoded tokens for each caption. The stand-alone code for data preprocessing is freely available from the DCASE 2020 audio captioning task and can be found on Github[2].

4.2 Hyper-parameters and training procedure

The development data split of Clotho is used to train and fine tune the hyper-parameters of the neural networks method. Due to limitations in computational resources, the batch size of 16 is employed in the training process. In order to apply a uniform T and S in a batch, vectors of zeros are prepended to to each $\mathbf{X}$, and vectors of ⟨eos⟩ are appended to every $\mathbf{Y}$ so that they have the same T and S equal

[2] `https://github.com/audio-captioning/clotho-dataset`

to the maximum T and S in the same batch. The method is trained for maximum of 1000 epochs. The binary cross-entropy loss function $\mathcal{L}(\hat{\mathbf{y}}_s, \mathbf{y}_s)$ is rounded to three decimal digits, and if it does not improve for 100 consecutive epochs, the training is stopped.

Moreover, it is observed that there is a noticeable imbalance in the appearing frequency of the tokens at the captions, for example, the words "a"/"an", or "the" have a high frequency, e.g. over 4000 times. However, some other tokens have much lower frequencies, e.g. five times. To overcome this problem, each token , denoted w_i, is inversely weighted by its frequency in the dataset. More specifically, a new loss formulation is used:

$$\mathcal{L}'(\hat{\mathbf{y}}_s, \mathbf{y}_s) = \Phi_s \mathcal{L}(\hat{\mathbf{y}}_s, \mathbf{y}_s), \tag{4.1}$$

where Φ_s is a weight for the loss calculation of the token represented by $\mathbf{y}_s$. However, with tokens with significant frequencies like "a", Φ_s could get values as low as $1e-5$, which is about 100 000 times lower compared to the weights of the lower frequencies tokens with $\Phi_s = 1$. This difference could significantly hamper the learning of the frequent tokens and could result in the loss of the frequent tokens to be zeros, since $\mathcal{L}'$ is rounded to three decimal digits. A solution employed is the clamping of Φ_s:

$$\Phi_s = \begin{cases} \frac{\min(f_w)}{f_{w_s}} & \text{if } \frac{\min(f_w)}{f_{w_s}} \geq \beta, \\ \beta & \text{otherwise} \end{cases}, \tag{4.2}$$

where $\min(f_w)$ is the minimum frequency of all tokens in the Clotho dataset, f_{w_s} is the frequency of the w_s token in the development split, β is a hyper-parameter called "clamp value frequency" and is set to $5e-1$, w_s is the token that $\mathbf{y}_s$ corresponds to.

Other hyper-parameters are according to the baseline method of DCASE2020: $L = 3$, $\Xi = 256$, and $\Psi = 256$. The parameters are optimized using the loss function $\mathcal{L}'$ and the Adam optimizer [54], with a learning rate of $1e-4$ and the same values for β_1 and β_2 according to the paper [54]. The dropout rate $p = 0.25$ is employed between $\text{RNN}^1_{\text{enc}}$ and $\text{RNN}^2_{\text{enc}}$, as well as between $\text{RNN}^2_{\text{enc}}$ and $\text{RNN}^3_{\text{enc}}$. To evaluate the effect of temporal sub-sampling, four different sub-sampling factors are used, such that $M = 2, 4, 8, 16$. The total number of parameters of the proposed method is 4 573 711, and the implementation code is based on the PyTorch framework.

4.3 Evaluation and metrics

To evaluate the method used in this thesis work, the metrics from DCASE 2020 audio captioning task are employed, using the evaluation split of the Clotho dataset. More specifically, these metrics consists of machine translation metrics BLEU_n, ROUGE_L, METEOR, and captioning metrics CIDEr, SPICE, SPIDEr. BLEU_n cal-

culates a weighted geometric mean of n-grams modified precision between predicted and ground truth captions [55]; ROUGE_L calculates an F-measure using a longest common sub-sequence (LCS) between predicted and ground truth captions [56]; METEOR computes a harmonic mean of the precision and recall for segments between predicted and ground truth captions [57]; CIDEr [58] uses term-frequency inverse-document-frequency (TF-IDF) weighting for n-grams to calculate a weighted cosine similarity between predicted and ground truth captions [58]; SPICE measures how well the predicted captions could recover from the ground truth captions [59]; SPIDEr is a weighted mean between CIDEr and SPICE to take advantages of both metrics [60]. Each metric is calculated between the predicted word sequence $\hat{\mathbf{Y}}$ and the ground truth word sequences $\mathbf{Y}$ for the same input $\mathbf{X}$.

5 Results and discussion

It can be seen from Table 5.1 that using a sub-sampling of factor $M \geq 2$ always improve the metrics of audio captioning methods. Sub-sampling with factor $M = 8$ yields the maximum value of the main metric SPIDEr at 0.067. However, increasing the value sub-sampling factor does not linearly improve the metrics produced. This could be because the method employs the fixed length vector z from the encoder, so the impact of reducing more length in the sequence could not be observed by the decoder. This indicates that the impact of increasing the sub-sampling factor could be more visible when using the whole output sequence of the encoder, e.g. the attention mechanism. However, more research needs to be carried out to verify this hypothesis.

Table 5.1 *Results for the baseline method, i.e. $M = 1$, and proposed method with sub-sampling factor $M = \{2, 4, 8, 16\}$.*

Metric	$\boldsymbol{M = 1}$	$\boldsymbol{M = 2}$	$\boldsymbol{M = 4}$	$\boldsymbol{M = 8}$	$\boldsymbol{M = 16}$
$BLEU_1$	0.389	**0.426**	0.418	0.417	**0.426**
$BLEU_2$	0.136	0.151	0.151	**0.154**	0.147
$BLEU_3$	0.055	0.058	0.061	**0.063**	0.058
$BLEU_4$	0.015	0.020	0.018	**0.025**	0.022
$ROUGE_L$	**0.262**	0.274	0.275	0.274	0.274
METEOR	0.084	**0.092**	0.091	0.089	0.090
CIDEr	0.074	0.092	0.090	**0.093**	**0.093**
SPICE	0.033	**0.040**	0.037	**0.040**	0.036
SPIDEr	0.054	0.066	0.064	**0.067**	0.064

In terms of time complexity, sequence temporal sub-sampling demonstrates a clear impact on the input sequence length and the duration needed for calculating predicted outputs using the Clotho evaluation data split, as shown in Table 5.2. With sub-sampling factor 2, the output sequence of the encoder is 75% less than the input sequence, the time needed to calculate all predicted captions also reduce 42% (from 58 sec to 34 sec). The value for SPIDEr in this case increase from 0.54 to 0.66. The time needed to train the same networks with sub-sampling factor 2 is only about 50% of the one with no sub-sampling (factor 1). The best sub-sampling on metrics SPIDEr is 8, which reduces the output sequence length by 98%, the inference time by 63%, with the training time at only about $\frac{1}{3}$ of the baseline model. Sub-sampling factor 16 also gives very promising results, which outputs the sequence length of only 5 to 10 time steps, SPIDEr metrics is a a bit less than sub-sampling 8, at 0.064. However, there is no significant difference in inference time between sub-sampling factor 8 and 16.

Table 5.2 *Reduction in length (in percentages), required time for predictions, minimum and maximum resulting number of time-steps (T_L^{min} and T_L^{max}, respectively), of sub-sampling factor $M = \{2, 4, 8, 16\}$ with $L = 3$.*

M	$T_L^{\min}$	$T_L^{\max}$	Reduction in length	Time (sec)
1	1292	2584	00.00%	58.81
2	323	646	75.00%	34.13
4	80	161	93.75%	25.67
8	20	40	98.43%	21.73
16	5	10	99.60%	20.42

Some examples of the final output of the sub-sampling encoder-decoder method with $M = 8$ is "*a person is walking a through something and*" for the audio file "clotho_file_01 A pug struggles to breathe 1_14_2008" in the Clotho evaluation split. Some of the ground truth captions for this audio file are "a small dog with a flat face snoring and groaning" and "a man walking who is blowing his nose hard and about to sneeze". Another example is the predicted caption "*a group of of birds and birds a*" compared to ground truth captions like "a flock of birds comes together with a lot of chirping" or "birds sing in different tones while in a large group" from the file "clotho_file_sparrows". Although the method manages to identify the objects and the actions of those objects, it lacks the language modelling to make the predicted captions resemble natural language, which is not the focus of this thesis.

6 Conclusions

This thesis examined an approach to audio captioning using deep neural networks, with the focus on solving the input-output length disparity problem, such that one word in a output caption corresponds to a much longer sequence of feature vectors in the input. The background knowledge for all concepts and techniques used to process the dataset, method construction and evaluation, as well as their implementations are reported.

The proposed method employs a multi-layered RNNs-based encoder and decoder architecture, in which the temporal sub-sampling is applied on the output of each layer in the encoder. Experimental results show that temporal sub-sampling can clearly improve the performance of the method, while reducing the input sequence length, inference time and training time for the system. The maximum improvement happens for the sub-sampling factor $M = 8$, where an increase of 0.013 is observed for the main metric SPIDEr, and inference time is reduced by 63%, compared to the same architecture without sub-sampling.

For future research, the temporal sub-sampling operation could be constructed to have its own weights, so it can learn to keep relevant input vectors, instead of only keeping the predefined vectors. Temporal sub-sampling could also be employed together with an alignment technique that takes the whole output of the encoder into account, which might make the effect of input sequence length reduction more noticeable.

References

[1] K. Drossos, S. Adavanne, and T. Virtanen, "Automated audio captioning with recurrent neural networks", in *2017 IEEE Workshop on Applications of Signal Processing to Audio and Acoustics (WASPAA)*, IEEE, 2017, pp. 374–378.

[2] K. Nguyen, K. Drossos, and T. Virtanen, *Temporal sub-sampling of audio feature sequences for automated audio captioning*, 2020. arXiv: 2007.02676 [eess.AS].

[3] R. Lyons, *Understanding Digital Signal Processing (3rd Edition)*. Aug. 2011, ISBN: 013702741-9.

[4] M. Müller, *Fundamentals of Music Processing*. Jan. 2015, ISBN: 978-3-319-21944-8. DOI: 10.1007/978-3-319-21945-5.

[5] R. Bracewell, "The fourier transform.", *Scientific American*, vol. 260 6, pp. 86–9, 92–5, 1989.

[6] D. Robinson and M. Hawksford, "Psychoacoustic models and non-linear human hearing", Jan. 2000.

[7] M. Sahidullah and G. Saha, "Design, analysis and experimental evaluation of block based transformation in mfcc computation for speaker recognition", *Speech Communication*, vol. 54, no. 4, pp. 543–565, 2012, ISSN: 0167-6393. DOI: https://doi.org/10.1016/j.specom.2011.11.004. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167639311001622.

[8] M. Xu, L.-Y. Duan, J. Cai, L.-T. Chia, C. Xu, and Q. Tian, "Hmm-based audio keyword generation", vol. 3333, Nov. 2004, pp. 566–574, ISBN: 978-3-540-23985-7. DOI: 10.1007/978-3-540-30543-9_71.

[9] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016, http://www.deeplearningbook.org.

[10] D. Choi, C. J. Shallue, Z. Nado, J. Lee, C. J. Maddison, and G. E. Dahl, *On empirical comparisons of optimizers for deep learning*, 2020. arXiv: 1910.05446 [cs.LG].

[11] D. P. Kingma and J. Ba, *Adam: A method for stochastic optimization*, 2017. arXiv: 1412.6980 [cs.LG].

[12] I. Sutskever, J. Martens, G. Dahl, and G. Hinton, "On the importance of initialization and momentum in deep learning", *30th International Conference on Machine Learning, ICML 2013*, pp. 1139–1147, Jan. 2013.

[13] J. Duchi, E. Hazan, and Y. Singer, "Adaptive subgradient methods for online learning and stochastic optimization", *Journal of Machine Learning Research*, vol. 12, pp. 2121–2159, Jul. 2011.

[14] K. Janocha and W. M. Czarnecki, *On loss functions for deep neural networks in classification*, 2017. arXiv: 1702.05659 [cs.LG].

[15] C. Nwankpa, W. Ijomah, A. Gachagan, and S. Marshall, *Activation functions: Comparison of trends in practice and research for deep learning*, 2018. arXiv: 1811.03378 [cs.LG].

[16] J. Feng and S. Lu, "Performance analysis of various activation functions in artificial neural networks", *Journal of Physics: Conference Series*, vol. 1237, p. 022 030, Jun. 2019. DOI: 10.1088/1742-6596/1237/2/022030.

[17] Y. Bengio, P. Lamblin, D. Popovici, H. Larochelle, and U. Montreal, "Greedy layer-wise training of deep networks", vol. 19, Jan. 2007.

[18] C. Couprie, C. Farabet, L. Najman, and Y. LeCun, *Indoor semantic segmentation using depth information*, 2013. arXiv: 1301.3572 [cs.CV].

[19] I. J. Goodfellow, Y. Bulatov, J. Ibarz, S. Arnoud, and V. Shet, *Multi-digit number recognition from street view imagery using deep convolutional neural networks*, 2014. arXiv: 1312.6082 [cs.CV].

[20] S. K. Kumar, *On weight initialization in deep neural networks*, 2017. arXiv: 1704.08863 [cs.LG].

[21] D. Mishkin and J. Matas, *All you need is a good init*, 2016. arXiv: 1511.06422 [cs.LG].

[22] S. Koturwar and S. Merchant, *Weight initialization of deep neural networks(dnns) using data statistics*, 2018. arXiv: 1710.10570 [cs.LG].

[23] D. Rumelhart, G. E. Hinton, and R. J. Williams, "Learning representations by back-propagating errors", *Nature*, vol. 323, pp. 533–536, 1986.

[24] E. Kreyszig, H. Kreyszig, and E. J. Norminton, *Advanced Engineering Mathematics*, Tenth. Hoboken, NJ: Wiley, 2011, p. 398, ISBN: 0470458364.

[25] S. Ruder, *An overview of gradient descent optimization algorithms*, 2016. arXiv: 1609.04747 [cs.LG].

[26] B. Li, E. Zhou, B. Huang, J. Duan, Y. Wang, N. Xu, J. Zhang, and H. Yang, "Large scale recurrent neural network on gpu", Jul. 2014, pp. 4062–4069. DOI: 10.1109/IJCNN.2014.6889433.

[27] M. Schuster and K. Paliwal, "Bidirectional recurrent neural networks", *Signal Processing, IEEE Transactions on*, vol. 45, pp. 2673–2681, Dec. 1997. DOI: 10.1109/78.650093.

[28] A. Graves, A. Mohamed, and G. Hinton, "Speech recognition with deep recurrent neural networks", in *2013 IEEE International Conference on Acoustics, Speech and Signal Processing*, 2013, pp. 6645–6649.

[29] E. Arisoy, A. Sethy, B. Ramabhadran, and S. Chen, "Bidirectional recurrent neural network language models for automatic speech recognition", in *2015 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, 2015, pp. 5421–5425.

[30] A. Graves, M. Liwicki, H. Bunke, J. Schmidhuber, and S. Fernández, "Unconstrained on-line handwriting recognition with recurrent neural networks", in *Advances in Neural Information Processing Systems 20*, J. C. Platt, D. Koller, Y. Singer, and S. T. Roweis, Eds., Curran Associates, Inc., 2008, pp. 577–584. [Online]. Available: `http://papers.nips.cc/paper/3213-unconstrained-on-line-handwriting-recognition-with-recurrent-neural-networks.pdf`.

[31] A. Graves and J. Schmidhuber, "Offline handwriting recognition with multidimensional recurrent neural networks", in *Advances in Neural Information Processing Systems 21*, D. Koller, D. Schuurmans, Y. Bengio, and L. Bottou, Eds., Curran Associates, Inc., 2009, pp. 545–552. [Online]. Available: `http://papers.nips.cc/paper/3449-offline-handwriting-recognition-with-multidimensional-recurrent-neural-networks.pdf`.

[32] Z. Xie, Z. Sun, L. Jin, Z. Feng, and S. Zhang, "Fully convolutional recurrent network for handwritten chinese text recognition", Dec. 2016, pp. 4011–4016. DOI: `10.1109/ICPR.2016.7900261`.

[33] K. Cho, B. van Merrienboer, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, *Learning phrase representations using rnn encoder-decoder for statistical machine translation*, 2014. arXiv: `1406.1078 [cs.CL]`.

[34] I. Sutskever, O. Vinyals, and Q. V. Le, *Sequence to sequence learning with neural networks*, 2014. arXiv: `1409.3215 [cs.CL]`.

[35] S. Hochreiter, "Untersuchungen zu dynamischen neuronalen netzen", Apr. 1991.

[36] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult", *IEEE transactions on neural networks / a publication of the IEEE Neural Networks Council*, vol. 5, pp. 157–66, Feb. 1994. DOI: `10.1109/72.279181`.

[37] R. Pascanu, T. Mikolov, and Y. Bengio, *On the difficulty of training recurrent neural networks*, 2013. arXiv: `1211.5063 [cs.LG]`.

[38] S. Hochreiter, "The vanishing gradient problem during learning recurrent neural nets and problem solutions", *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, vol. 6, pp. 107–116, Apr. 1998. DOI: 10.1142/S0218488598000094.

[39] S. Hochreiter and J. Schmidhuber, "Long short-term memory", *Neural computation*, vol. 9, pp. 1735–80, Dec. 1997. DOI: 10.1162/neco.1997.9.8.1735.

[40] K. Drossos, S. Lipping, and T. Virtanen, *Clotho: An audio captioning dataset*, 2019. arXiv: 1910.09387 [cs.SD].

[41] F. Scheidegger, L. Cavigelli, M. Schaffner, A. C. I. Malossi, C. Bekas, and L. Benini, "Impact of temporal subsampling on accuracy and performance in practical video classification", in *2017 25th European Signal Processing Conference (EUSIPCO)*, 2017, pp. 996–1000.

[42] S. Lipping, K. Drossos, and T. Virtanen, "Crowdsourcing a dataset of audio captions", *arXiv preprint arXiv:1907.09238*, 2019.

[43] Y. Koizumi, D. Takeuchi, Y. Ohishi, N. Harada, and K. Kashino, *The ntt dcase2020 challenge task 6 system: Automated audio captioning with keywords and sentence length estimation*, 2020. arXiv: 2007.00225 [eess.AS].

[44] Y. Wu, K. Chen, Z. Wang, X. Zhang, F. Nian, S. Li, and X. Shao, "Audio captioning based on transformer and pre-training for 2020 dcase audio captioning challenge technical report", 2020.

[45] Q. Kong, Y. Cao, T. Iqbal, Y. Wang, W. Wang, and M. D. Plumbley, *Panns: Large-scale pretrained audio neural networks for audio pattern recognition*, 2020. arXiv: 1912.10211 [cs.SD].

[46] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, *Attention is all you need*, 2017. arXiv: 1706.03762 [cs.CL].

[47] H. Wang, B. Yang, Y. Zou1, and D. Chong, *Automated audio captioning with temporal attention*, 2020. [Online]. Available: http://dcase.community/documents/challenge2020/technical_reports/DCASE2020_Wang_5_t6.pdf.

[48] X. Xu, H. Dinkel, M. Wu, and K. Yu, "The sjtu submission for dcase2020 task 6: A crnn-gru based reinforcement learning approach to audiocaption technical report", 2020.

[49] S. Perez-Castanos, J. Naranjo-Alcazar, P. Zuccarello, and M. Cobos, *Listen carefully and tell: An audio captioning system based on residual learning and gammatone audio representation*, 2020. arXiv: 2006.15406 [eess.AS].

[50] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, ser. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2012, ISBN: 9783642247972. [Online]. Available: `https://books.google.fi/books?id=wpb-CAAAQBAJ`.

[51] F. Font, G. Roma, and X. Serra, "Freesound technical demo", in *Proceedings of the 21st ACM international conference on Multimedia*, 2013, pp. 411–412.

[52] K. Crowston, "Amazon mechanical turk: A research tool for organizations and information systems scholars", in *Shaping the Future of ICT Research. Methods and Approaches*, A. Bhattacherjee and B. Fitzgerald, Eds., Berlin, Heidelberg: Springer Berlin Heidelberg, 2012, pp. 210–221, ISBN: 978-3-642-35142-6.

[53] K. Sechidis, G. Tsoumakas, and I. Vlahavas, "On the stratification of multi-label data", in *Joint European Conference on Machine Learning and Knowledge Discovery in Databases*, Springer, 2011, pp. 145–158.

[54] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization", in *Proceedings of the International Conference on Learning Representation (ICLR)*, 2014.

[55] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu, "Bleu: A method for automatic evaluation of machine translation", in *Proceedings of the 40th annual meeting on association for computational linguistics*, Association for Computational Linguistics, 2002, pp. 311–318.

[56] C.-Y. Lin, "ROUGE: A package for automatic evaluation of summaries", in *Text Summarization Branches Out*, Barcelona, Spain: Association for Computational Linguistics, Jul. 2004, pp. 74–81. [Online]. Available: `https://www.aclweb.org/anthology/W04-1013`.

[57] A. Lavie and A. Agarwal, "Meteor: An automatic metric for mt evaluation with high levels of correlation with human judgments", in *Proceedings of the second workshop on statistical machine translation*, 2007, pp. 228–231.

[58] R. Vedantam, C. Lawrence Zitnick, and D. Parikh, "CIDEr: Consensus-based image description evaluation", in *Proceedings of the IEEE conference on computer vision and pattern recognition (CVPR)*, 2015, pp. 4566–4575.

[59] P. Anderson, B. Fernando, M. Johnson, and S. Gould, "Spice: Semantic propositional image caption evaluation", in *European Conference on Computer Vision*, Springer, 2016, pp. 382–398.

[60] S. Liu, Z. Zhu, N. Ye, S. Guadarrama, and K. Murphy, "Improved image captioning via policy gradient optimization of spider", in *2017 IEEE International Conference on Computer Vision (ICCV)*, 2017, pp. 873–881.